A *Database Management System (DBMS)*, the broader area of this research, is a collection of interrelated, interdependent and interacting software modules that manage a *database*. A database is a collection of interrelated organized data. *Indexing* is an integral part of any database that facilitates faster query processing and response. *R*-tree [1] is an indexing technique which is used for indexing the data that belongs to *multidimensional object sets*. As part of the research, several ameliorations were carried out on R*-tree principles to enhance the overall performance of a database system. An overview of the placement of the research realm within DBMS is given in Figure 1.1.

![Diagram of Database Management System](image)

**Figure 1.1 Overview of the research realm**

1.1 **Meaning of Dimensions in Databases**

The term *dimension* has a unique connotation in DBMS which is different from the conventional connotation. Theoretically, the dimensions that are encountered in a
DBMS range from zero to infinity. The rudimentary and ramifications of dimensions in DBMS are explained below.

1.1.1 Objects and Object Sets

An object is anything that can be conceptually differentiated from the rest of the things in the universe. The basic differentiation is made using attributes. All the objects that have the same set of attributes belong to the same object set. The objects within an object set are differentiated using attribute values. The possible set of values for an attribute is called the domain of the attribute. For example, all the cities of the world form an object set Cities, since all cities have the same set of attributes, Latitude, Longitude and NameOfCity. But within this object set, the objects, that is, the cities are differentiated using the attribute values which are the actual latitude values, longitude values and the names of the cities. The domain of the Latitude is the set of all possible latitudes, domain of Longitude is the set of all possible longitudes and domain of NameOfCity is the set of all the cities in the world.

All the attributes of an object set need not participate in identifying the objects uniquely. The minimal combination of attributes that is required to uniquely identify an object in an object set is called as the primary key or key of the object set. Any other combination is called as secondary key. Secondary keys are beyond the scope of this research. A key that has only one attribute is called as single attribute key. A key that has multiple attributes is called multiple attribute key or concatenated key or composite key.

1.1.2 Multidimensional Objects and Multidimensional Object Sets

An object with a single key is called as a single dimensional object. An object with a concatenated key is called as a multidimensional object. A set of single dimensional objects is called single dimensional object set. A set of multidimensional objects is called multidimensional object set.

Consider the object set, Cities, discussed in preceding section. Cities is a 2-dimensional object set, since Latitude and Longitude form the key. Analogously,
students in a class form a 1-dimensional object set *Students*, since a single attribute *Student#* constitutes the key; reservations for seats in an airplane forms a 3-dimensional object set *Reservations*, since the combination of the attributes *Flight#*, *Seat#* and *DateOfTravel* constitute the key; sales details of a salesman forms a 4-dimensional object set *SalesDetails*, since the combination of the attributes *Salesman#*, *Product#*, *Area#* and *DateOfSale* constitute the key. Higher dimensionalities are defined similarly.

Multidimensional objects and objects sets are abundant in application areas like multimedia, entertainment, image processing and recognition, geographic information system, weather forecasting and remote sensing. The dimensionality of objects in these application areas normally range from one to around twenty. Currently, dimensionalities of more than twenty have more research relevance than application relevance.

### 1.1.3 Abstraction of Multidimensional Objects and Object Sets

A key that has *d* attributes forms a *d*-dimensional space with *d* axes, *D*₁, *D*₂, ..., *D*ₙ. A set of values for the combination of key attributes is abstracted as a point, that is, a 0-dimensional object in a *d*-dimensional space. Each object in the *Students* object set discussed in preceding section is a 0-dimensional point in 1-dimensional space; each object in *Cities* object set is a 0-dimensional point in 2-dimensional space; each object in *Reservations* object set is a 0-dimensional point in 3-dimensional space; each object in *SalesDetails* object set is a 0-dimensional point in 4-dimensional space. Objects in higher dimensionalities are similarly abstracted.

Several corollaries can be derived from the abstraction. The dimensionality of objects in *d*-dimensional space may range from 0 to *d*. Two points (*a₁*, *a₂*, ..., *aₙ*) and (*b₁*, *b₂*, ..., *bₙ*) form the diagonal of a *d*-dimensional cube. *a₁*, *b₁* ∈ domain(*D*₁); *a₂*, *b₂* ∈ domain(*D*₂) and *aₙ*, *bₙ* ∈ domain(*D*ₙ). domain(*Dᵢ*) is the domain of the attribute represented by axis *Dᵢ*. A point (*c₁*, *c₂*, ..., *cₙ*) and radius *r* form a *d*-dimensional sphere. *cᵢ* ∈ domain(*Dᵢ*). In general, it is possible to define any *d*-dimensional volumes in *d*-dimensional space. Moreover, it is also possible to have an aggregate object with many components, each one varying in its dimensionality from the other.
Due the abstraction described above, multidimensional objects are also referred as *spatial* objects. Since space and time are complementary to each other, the multidimensional objects are also treated as *temporal* objects for certain applications and domains. *Spatio-temporal* is another higher level abstraction that is widely in existence and extensively studied.

1.1.4 Multidimensional Database System

A *multidimensional database* is an organized collection of interrelated data that belongs to multidimensional objects and object sets. A *multidimensional database system* is a collection of interrelated, interdependent and interacting software modules that manage a multidimensional database.

Every database is pivoted upon a key concept called *database model* around which every component of the database is built. A database model has in its core the *data structure* into which every data has to be accommodated. A database language is defined in tandem with the data structure to facilitate the construction of queries with the data structure. For example, in *Relational* data base model, the data structure is the *table* and the database language is *Relational Algebra* and *Relational Calculus*. A lot of other database models are in vogue. Some are application domain generic and others are application domain specific. Multidimensionality encompasses all the models. In this thesis, Relational model has been adopted as the reference model for illustrations and explanations. Hence, the *Cities* object set in previous section can be construed as a *table* with three columns. It has a concatenated key *Latitude+Longitude*. *Reservations* object set can be construed as a *table* with three columns and has a key *Flight#+*Seat#+*DateOfTravel*. *SalesDetails* object set can be construed as a *table* with four columns and has a key combining all the columns of the table. Each set of data that belongs to an object corresponding to table’s object set is called a *record* or a *row* or a *tuple*.

In this thesis the term record, row, tuple and object are used interchangeably. Similarly, the terms table and object sets are also used interchangeably. Records are also referred to as points and tables as object spaces in appropriate places.
1.2 Database Organization

The two primitive operations that are performed during query processing are reading and writing. The three database events, namely insert, delete and update are sequences of read and write operations. The word database connotes that, the volume of data to be processed is large and is available in a peripheral storage medium. Entire data cannot be brought into the core memory at the same time. Only a negligible portion of the data can be brought into the core memory at any given point of time and hence the completion of the database operations requires large number of peripheral storage input-output operations. But the peripheral storage input-output operations are one of the slowest operations in a system and hence the performance of a database depends upon the parameters of these devices. Database organization refers to the ways the data of the object set are stored and retrieved in these peripheral storage devices. Database organization affects the queries at every stage of its processing and execution.

1.2.1 Types of Database Organization

There are three basic types of organization for data in the database namely, sequential, indexed and random. In sequential organization, data are read in the order in which they were stored. The order of the data is fixed. New data are added at the end. In indexed organization a Meta structure called the index contains key data and pointer to the location of where the data of the entire object is stored. Access to the data of the object is performed only through the index. The organization of the index determines the performance of the database. In random organization, data are organized in buckets to which the data are mapped. Mapping is carried out using well defined mathematical functions. The process of applying the mathematical function on the key data for storing and retrieving data is called hashing. This research pertains to indexing. Sequential and random organizations are beyond the scope of this research.

1.2.2 Objective of Database Organization

The main aim of database organization is to store and retrieve data in lesser time. The major factors that decide the database organization in a system are:
a) Storage device and  
b) Application.

In some devices like tapes, only sequential organization is possible, whereas in devices like hard disks, indexed and direct organizations are possible. While payroll kinds of applications prefer sequential organization, interactive applications prefer indexed and random organization. Hence, the choice of the organization in the context of the objective has to be done meticulously for better performance of the database.

1.2.3 Indexed Organization

A database system works towards achieving several objectives. One of the primary objectives is to respond to queries at a faster rate. This can be achieved by reducing the number of peripheral storage input-output operations. *Indexes* facilitate the reduction. An *index* may be described as any extraneous structure apart from the actual data store that facilitates storage, retrieval, search and analysis of data across classifications and categories without retrieving the complete data store. Consequently, *indexing* is any efficient method consisting of both structure and process that facilitates storage, retrieval, search and analysis of data across classifications and categories without retrieving the complete data store. An illustrative data store and the corresponding index structure are shown in Figure 1.2. The following discussion illustrates the reduction of peripheral storage input-output operations using indexes.

Assume that each entry in the data store, that is, the table takes 100 bytes. Hence, the total size of the table is 1600 bytes. Assume that each index entry takes 20 bytes. The total size of the index is 320 bytes. Assume that a buffer of size 100 bytes is allocated for data transfer operations. Assume that the data of student with *Student# = 'S30' is searched. Then without using index, that is sequentially searching, 15 input-output operations will have to be done. Using index, the same response can be achieved using 5 input-output operations, 4 for retrieving the index and 1 for the retrieving the actual data from the data store.
An index that is constructed on a single attribute is called a \textit{single dimensional index}. An index that is constructed on multiple attributes is called a \textit{multidimensional index}.

![Figure 1.2 A data store and the corresponding index](image)

1.2.4 \textbf{Hierarchical Index}

In some cases indexes are maintained in sequential order. In such cases a combination of sequential and indexed organizations is adopted for organization. Such organization is illustrated in Figure 1.3. The index contains one entry for a certain number of rows, except the last level. The index assists in reaching the desired group of rows in less number of operations. Within the group the rows are searched sequentially. This organization is called \textit{indexed sequential organization}. In this organization, the index itself is a multilevel tree shaped index or in other words \textit{hierarchical index}. This research pertains to such types of hierarchical indexes.
1.3 Multidimensional Indexing and Partitioning of Data

A single dimensional hierarchical index basically collects data into groups and provides paths to the groups that are traversed during the index operations. The data in the groups maintain the natural predecessor-successor relationship of values and hence can be called as linear groups. In d-dimensional multidimensional hierarchical indexes, extends this idea to higher dimensions. But the result is not linear groups but d-dimensional groups that take the shape of d-dimensional rectangles. A d-dimensional
rectangle is iso-oriented and is bounded by \( d-1 \) dimensional objects. This bounding is recursive in lower dimensions.

If a multidimensional indexing method forms groups by partitioning the data space, then the method is called space partitioned multidimensional indexing method. On the other hand, if a multidimensional indexing method forms groups by partitioning the data then the method is called data partitioned multidimensional indexing method. In space partitioned method, the groups are disjoint but collectively cover the entire object space at every level of the hierarchy. In data partitioned method, groups overlap but collectively cover the entire object space at every level of the hierarchy.

1.4 Multidimensional Queries

The abstraction of multidimensional objects and object sets introduce many relationships between the objects and hence many types of queries. These queries are classified based on whether the user defines a region of the data space or an intended size of the result. Multidimensional queries are also referred as spatial queries in this thesis. The major types of multidimensional queries are:

a) Range Queries,

b) Nearest Neighbor Queries (NNQs) and

c) Join Queries.

1.4.1 Range Query

In a range query [2], a query point \( Q \), a distance \( r \), and a metric \( M \) are specified. The result set comprises all points from the space, which have a distance smaller or equal to \( r \) from \( Q \) according to metric \( M \). Point queries are range queries with a radius \( r = 0 \) and an arbitrary metric \( M \). Window query, a special type of range query, specifies a rectangular region in data space, from which all points in the space are selected. The specified hyper-rectangle is always parallel to the axis and hence called as a window.

Yet another special type of the range query is the range aggregate query. Specifically, given a set \( S \) of points in the \( d \)-dimensional space, a range aggregate query returns a single value that summarizes the set \( R \subset S \) of points in a \( d \)-dimensional hyper-
rectangle \( q \) according to some aggregation function [3]. Apart from the ramifications of the range query mentioned above, there are some special categories of queries that come under range queries, viz., similarity search, directional and topological.

In similarity search query [4] the degree of similarity between two objects is measured by some distance function between their feature vectors. The search returns the objects that are nearest to the query object in high-dimensional spaces. In directional query [5] objects are retrieved based on the four 2-dimensional directions: north, east, west and south. In topological query [6] objects are retrieved using topological relationships. Topological relationships are the ones that stay invariant under topological transformations such as translation, rotation and scaling.

1.4.2 Nearest Neighbor Query

The classical NNQ [2] returns exactly one point object as result, the object with the lowest distance to the query point among all points stored in the database. A variation of the NNQ is the \( k \)-nearest neighbor query (kNNQ). The kNNQ selects \( k \) points from the space such that no point among the remaining points in the space is closer to the query point than any of the selected points.

A variant of kNNQs is the Ranking Query which does not require that the user specifies a range in the data space or a result set size. The first answer of a ranking query is always the NN. Then upon further requests, the second NN is reported, then the third and so on. In approximate NNQs only the points which are not much farther away from the query point than the exact NNs are retrieved.

Yet another variant of the NNQ is the reverse NNQ. Given an arbitrary query point \( q \), this operation retrieves all points of the space to which \( q \) is the NN. A recent extension of NNQs is the closest pair query which is also called the distance join query.

1.4.3 Join Query

A join query [7, 8] finds pairs of data points satisfying a particular property. Given two \( d \)-dimensional data sets \( B \) and \( R \), the similarity join query searches for the pairs of points \( p = (p_1, ..., p_d) \) from \( B \) and \( q = (q_1, ..., q_d) \) from \( R \) such that the distance
between them is less than an input value $e$, according to the metric of interest. The top-$k$ closest pairs join query, instead, searches for the $k$ pairs from the two data sets $B$ and $R$ having the $k$ smallest distances between them.

1.5 Principles of Indexing

The effectiveness of every query execution process is affected by several principles associated with indexes. Amelioration made on the ramifications of these principles eventually result in better performance of the database system as a whole. The nuances in principles are sensitive to query categories. Hence, subjective ameliorations have to be made to achieve better performances. The major principles are:

a) Representations of the objects,
b) Data structures of the objects,
c) Storage and retrieval process,
d) Cost estimation of queries,
e) Storage in the physical medium and
f) Post implementation fine tuning.

These principles are hinged on the indexing structure and method involved in the query processing.

1.6 R*-tree

The R*-tree [1] is an improved model of the variants of R-tree [9]. It forms the basis of all the multidimensional indexing methods based on data partitioning that are in existence today. R*-tree is a hierarchical, dynamic indexing structure. A sample 2-dimensional data set and the corresponding R*-tree is shown in Figure 1.4. If $d$ is the dimensionality of the space whose objects are indexed, then R*-tree uses a $d$-dimensional Minimum Bounding Rectangles (MBRs) to cover the objects. In Figure 1.4, MBR with identity R3 demonstrates the coverage of an object by a MBR. These MBRs are grouped together in leaf nodes according to their spatial proximity, which are then recursively grouped in higher levels up to the root.
Every leaf node of the R*-tree contains between $m$ and $M$ index records unless it is the root. $M$ is the order of the R*-tree that specifies the maximum number of entries that will be fit in one node and $m = M/2$. For each index record $(I, tuple-identifier)$ in a leaf node, $I$ is the smallest rectangle that spatially contains the $d$-dimensional data object represented by the indicated tuple. In every non-leaf entry $(I, child-pointer)$ in a non-leaf node, $I$ is the smallest rectangle that spatially contains the rectangles in the child node. Non-leaf MBRs are also called as directory rectangles. The root node has at least two children unless it is a leaf. All leaves appear on the same level. ‘tuple-identifier’ refers to a tuple in a database and $I$ is $d$-dimensional rectangle which is the bounding object of spatial object indexed $I = (I_0, I_1, ..., I_{d-1})$. MBRs as well as partitions of R*-trees overlap each other. The space occupied by a node is called as node region.

During the process of insertion and deletion the objectives that a R*-tree aims at fulfilling are:

a) Minimization of the area covered by directory rectangles,

b) Minimization of the overlap between directory rectangles,

c) Minimization of the margin of directory rectangles,
d) Maximization of storage utilization and
e) Minimization of the number of node splits.

The first four objectives are achieved by
i. Choosing a split-axis, the axis along which the splitting is best and
ii. Calculating the optimal distribution along the split-axis.

The last objective is achieved by forced reinsertion.

1.7 Scope and Objectives of the Research

A multidimensional query processing passes through various stages before producing the desired output. A pictorial presentation of various stages and modules involved in the processing of multidimensional queries using R*-tree indexing is given in Figure 1.5.

If the multidimensional query submitted involves insert, delete or update operations, the storage strategy manager forms an efficient storage strategy for storage. If the multidimensional query submitted involves select operations, the execution strategy manager forms an optimal execution strategy before execution. In some cases both managers interact to form the strategies. The objective of every strategy formation is to effect less number of input-output operations on the R*-trees.

The overall objective of this research is to study and ameliorate the R*-tree principles that would assist in improved multidimensional query processing.

Out of the several problems identifiable pertaining to the principles with regard to R*-tree, six were chosen for amelioration as part of the research. They are:

a) to develop an improved method to estimate the number of node accesses in an R*-tree for window queries,
b) to develop an improved method to estimate the number of node accesses in an R*-tree for kNNQs,
c) to develop an improved representation scheme and an algorithm to retrieve multidimensional objects based on directional and topological relationships (DaTRs) using R*-trees. Subsequently, estimate the number of node
accesses in the R*-tree for the given directional and topological queries (DaTQs),

d) to develop a method to efficiently organize R*-trees in secondary storage mediums such as hard disks,

---

Figure 1.5 Overview of multidimensional query processing using R*-tree
e) to develop an efficient algorithm for re-indexing R*-trees and
f) to use Minimum Bounding Polygons (MBPs) for objects instead of MBRs and Minimum Bounding Spheres (MBSs) to improve the performance of R*-tree.

1.8 Summary of Research Contributions

In this research work, several areas were explored to ameliorate the performance of R*-tree. The areas that were explored and the contributions made are explained below.

Estimation of the number of node accesses in an R*-tree for window queries

In database systems, the initial step in the formation of the strategy for query execution is the estimation of various parameters involved. One of the most important among these parameters is the number of nodes that would be accessed in indexing structures of the tables involved. This work pertains to the estimation of the number of nodes that would be accessed in a single R*-tree for window queries. Among diversified approaches available in the literature, methods based on the estimation of MBR extents have gained momentum due to their broad applicability. But these methods consider only the length distribution of the data set and the query in their estimations. Location distribution was assumed to be uniform, which is not true in most of the data sets. Hence, they do not provide accurate results. In this work, apart from length distribution, the actual location distribution was also considered in estimating the number of nodes that would be retrieved from the R*-tree for a given window query.

Estimation of the number of node accesses in an R*-tree for k-nearest neighbor queries

Estimating the number of nodes of the indexes that would be accessed for a given query is an important part of strategy formation for faster query execution. This work pertains to the estimation of the number of nodes that would be accessed in an R*-tree for a given kNNQ. The methods in the literature give good estimations when $k$ is
large, such as 1000 or above. For smaller values of $k$, the estimations are unacceptably inaccurate. In this work, theoretical analysis was carried out to study the reasons for the failure of the existing models when $k$ is small. The findings of the analysis were incorporated into the new method for accurate estimation of the said parameter even when $k$ takes smaller values, such as 10.

**Retrieval of multidimensional objects based on directional and topological relationships using $R^*$-trees and the estimation of the number of node accesses for such queries**

This work pertains to the study and improvement of data structures and algorithms for the retrieval of multidimensional objects based on DaTQs. Two key issues of this context are:

a) formulation of the representation, that is, the data structure of the DaT queries in multiple dimensions and

b) formulation of the algorithm that works in tandem with the representation.

In this work the following were developed.

i. an efficient representation scheme was designed for the DaT queries in multiple dimensions and

ii. an algorithm that works in tandem with the deliberated representation scheme was developed.

Subsequently, a model for the estimation of the node accesses based on the proposed method was also developed.

**Efficient physical organization of $R^*$-trees**

A number of attempts have been made in the past to improve the performance of $R^*$-tree by manipulating the tree parameters and the data parameters. But hardly few attempts had been made to use external parameters such as disk parameters along with the internal parameters to enhance its performance. This work pertains to physical organization of $R^*$-trees. Peripheral storage mediums such as hard disks are the places where the $R^*$-trees are ultimately stored into and retrieved from. If an efficient mapping
can be performed between the key units of the logical structure and the physical medium, improvement in performance can be achieved. In this work, the nodes, which are the key units of the R*-trees were efficiently clustered into the sectors, the key units of the peripheral storage medium. Clustering was performed within the constraint that the independence between the logical and physical organization of the R*-tree should be preserved. Moreover, to preserve the structural and functional properties of R*-tree at any point in the process of clustering, a concept called controlled duplication was employed.

Re-indexing R*-trees

There are many routine database maintenance activities that are performed to sustain the database system at its peak performance. One such activity is the maintenance of the indexing structures. This work pertains to re-indexing, one of the principal maintenance activities that has to be performed for efficient query processing. When insertion, deletion and updating are performed dynamically, indexes become fragmented. Apart from fragmentation of the indexes, multidimensional indexing structures also suffer from empty spaces. In this work, a new method has been devised to re-index R*-trees. The method recursively regroups the nodes from the leaves in such a way that the above said problems are minimized.

Indexing using minimum bounding polygons

R*-trees use MBRs to represent spatial objects. MBRs are easier to represent, store and retrieve. The associated algorithms are also simpler and less time consuming. But the major drawback with MBRs is that, they do not represent the real objects accurately in most cases and hence introduce empty space with objects. Empty spaces are detrimental to the performance of R*-trees. This work pertains to the exploration of alternate structures to represent real life objects. In the literature the possibility of MBSs and combination of MBRs and MBSs had been explored. In this work, the prospects of MBPs instead of MBSs and MBRs were explored.
1.9 **Organization of Chapters in the Thesis**

The thesis consists of nine chapters. The contents of each chapter are given below in brief.

Chapter 2 presents the literature survey of the research domain to enumerate the work that had been carried out in the past.

Chapter 3 depicts a method to estimate the number of nodes accessed in an R*-tree for window queries. The efficiency of the method is ascertained by comparing it with other existing methods.

Chapter 4 deals with a method to estimate the number of nodes accessed in an R*-tree for kNNQs. The efficiency of the method is theoretically and experimentally proved.

Chapter 5 presents a method with improved representation scheme and an algorithm that works in tandem with the representation for the retrieval of multidimensional objects based on DaTRs using R*-trees. A new estimation model for the number of node accesses for a given DaTQ has been developed. The method and the model are compared with the existing ones and the experimental results are presented to ascertain their superiority.

Chapter 6 illustrates a method to efficiently store R*-trees in the peripheral storage medium. The method does not compromise the independence between the logical and physical organization of R*-trees. Efficiency of the method presented is theoretically explained and experimentally enumerated.

Chapter 7 proposes a method to re-index R*-trees. Experimental results that ascertain the performance fine tuning are also presented.

Chapter 8 illustrates a method to use MBPs to represent the real life objects instead of MBRs and MBSs. The advantages of using MBPs are enumerated. Experimental results that ascertain the effectiveness of MBPs are also presented.

Chapter 9 concludes the thesis and outlines the sequel to the research.